Casos Prácticos

La **búsqueda binaria** es un ejemplo clásico del paradigma de divide y vencerás. Consiste en dividir el problema en subproblemas más pequeños y resuelve cada uno de ellos de forma recursiva, mejorando la optimización y escalabilidad de proyectos. (Niloufar Shafiei, s.f.)

En la búsqueda binaria, el array de entrada está ordenado, ya que de otra forma no funcionaria, debido a que se basa en la premisa de que al comparar un elemento central con la clave de búsqueda, se puede determinar si la clave está en la mitad izquierda o derecha del array. Si el array no está ordenado, esta premisa no se sostiene, y no se puede garantizar que los elementos a la izquierda del elemento central sean todos menores que él y los de la derecha sean todos mayores. Como alternativa, se puede ordenar el array primero utilizando un algoritmo de ordenamiento como **QuickSort**, **MergeSort** o **HeapSort**.

El algoritmo funciona dividiendo el array en dos partes en cada paso. Luego, compara la clave de búsqueda con el elemento central:

- Si la clave es menor, se continúa buscando en la mitad izquierda.
- Si la clave es mayor, se busca en la mitad derecha.

Este proceso de dividir a la mitad se repite hasta que se encuentra la clave o se concluye que no está presente.

Eficiencia de usar búsquedas binarias

Cada vez que se divide el array en dos, el tamaño del problema se reduce a la mitad. Por lo tanto, el número de divisiones que se necesitan para llegar a una sola posición es aproximadamente el logaritmo en base 2 del número de elementos.

Por ejemplo:

- Si el array tiene 16 elementos, la búsqueda binaria requerirá aproximadamente log2(16)=4log2(16)=4 comparaciones.

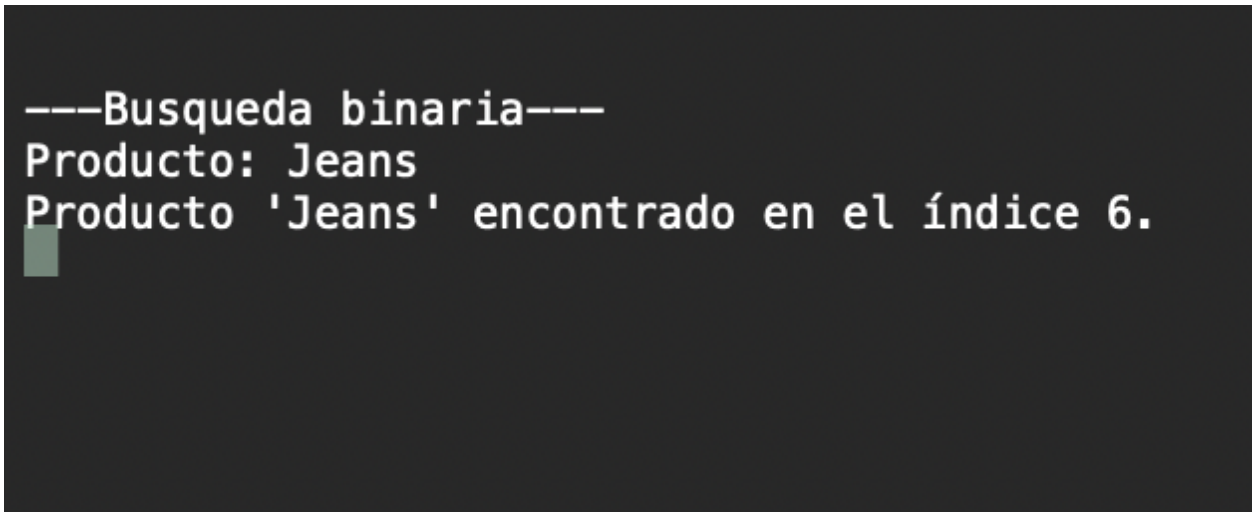
- Si el array tiene 1024 elementos, se necesitarán log2(1024)=10log2(1024) =10 comparaciones.

En general, si el array tiene n elementos, el número máximo de pasos será aproximadamente log2(n)log2(n), lo que significa que el tiempo de ejecución de la búsqueda binaria es O(logn)O(logn). Esto es mucho más eficiente que una búsqueda lineal, que tiene un tiempo de ejecución de O(n)O(n), ya que la búsqueda lineal inspecciona cada elemento uno por uno.

Ejemplo de caso en C Sharp

```
class Programa
{
    // Método de búsqueda binaria
    static int BusquedaBinaria(string[] productos, string objetivo)
    {
        int inicio = 0;
        int fin = productos.Length - 1; //Numero de elementos del arreglo

        while (inicio <= fin)
        {
            int medio = inicio + (fin - inicio) / 2;
            //Divide la longitud entre 2 para encontrar la mitad del rango

            // Compara el elemento en la posición media con el objetivo
            int comparacion = String.Compare(productos[medio], objetivo, StringComparison.OrdinalIgnoreCase);

            if (comparacion == 0)
            {
                return medio; // Retorna el índice si el objetivo es encontrado
            }
            else if (comparacion < 0)
            {
                inicio = medio + 1; // Busca en la mitad derecha
            }
            else
            {
                fin = medio - 1; // Busca en la mitad izquierda
            }
        }

        return -1; // Retorna -1 si el objetivo no es encontrado
    }
```

```
static void Main()
{
    // Array de productos de ropa ordenados alfabéticamente
    string[] productosDeRopa =
    {
        "Blusa", "Camisa", "Chaqueta", "Cinturón", "Falda",
        "Gorra", "Jeans", "Pantalones", "Sombrero", "Suéter",
        "Vestido", "Zapatos"
    };

    // Producto que queremos buscar
    Console.WriteLine("---Busqueda binaria---");
    Console.Write("Producto: ");
    string productoBuscado = Console.ReadLine();


    // Realiza la búsqueda binaria
    int resultado = BusquedaBinaria(productosDeRopa, productoBuscado);

    if (resultado != -1)
    {
        Console.WriteLine($"Producto '{productoBuscado}' encontrado en el índice {resultado}.");
    }
    else
    {
        Console.WriteLine($"Producto '{productoBuscado}' no encontrado.");
    }
    Console.ReadKey();
}
```



Explicación

El código implementa **búsqueda binaria** para encontrar un producto en un array de ropa ordenado alfabéticamente.

1. **Inicialización**: Se definen los límites de búsqueda (**inicio y fin**), siendo **inicio = 0** y **fin = productos.Length - 1.**
2. **Bucle while**: Mientras **inicio** sea menor o igual a **fin**, se calcula el **índice medio**: **“int medio = inicio + (fin - inicio) / 2;”**

Luego, comparamos el producto en productos[medio] con el producto buscado (objetivo):

- Si son iguales, se retorna el **“índice medio”.**
- Si “**objetivo”** es mayor, se ajusta inicio a “**medio + 1 (mitad derecha)”**
- Si “**objetivo”** es menor, se ajusta fin **a “medio - 1 (mitad izquierda)”**

1. **Resultado**: Si encuentra el producto, retorna su índice. Si no, retorna **-1**, indicando que el producto no está en el array. hola hola